World Wide Web: Internet and Web Information Systems, 8, 101–126, 2005

Query-Free News Search

MONIKA HENZINGER, BAY-WEI CHANG, BRIAN MILCH and SERGEY BRIN bay@google.com
Google Inc., 1600 Amphitheatre Parkway, Mountain View, CA 94043, USA

Online version published in December 2004

Abstract

Many daily activities present information in the form of a stream of text, and often people can benefit from additional information on the topic discussed. TV broadcast news can be treated as one such stream of text; in this paper we discuss finding news articles on the web that are relevant to news currently being broadcast.

We evaluated a variety of algorithms for this problem, looking at the impact of inverse document frequency, stemming, compounds, history, and query length on the relevance and coverage of news articles returned in real time during a broadcast. We also evaluated several postprocessing techniques for improving the precision, including reranking using additional terms, reranking by document similarity, and filtering on document similarity. For the best algorithm, 84–91% of the articles found were relevant, with at least 64% of the articles being on the exact topic of the broadcast. In addition, a relevant article was found for at least 70% of the topics.

Keywords: query-free search, web information retrieval

1. Introduction

Many daily activities present information using a written or spoken stream of words: television, radio, telephone calls, meetings, face-to-face conversations with others. Often people can benefit from additional information about the topics that are being discussed. Supplementing television broadcasts is particularly attractive because of the passive nature of TV watching. Interaction is severely constrained, usually limited to just changing the channel; there is no way to more finely direct what kind of information will be presented.

Indeed, several companies have explored suggesting web pages to viewers as they watch TV. For example, the *Intercast system*, developed by Intel, allows entire HTML pages to be broadcast in unused portions of the TV signal. A user watching TV on a computer with a compatible TV tuner card can then view these pages, even without an Internet connection. NBC transmitted pages via Intercast during their coverage of the 1996 Summer Olympics. The *Interactive TV Links system*, developed by VITAC (a closed captioning company) and WebTV (now a division of Microsoft), broadcasts URLs in an alternative data channel interleaved with closed caption data [6,18]. When a WebTV box detects one of these URLs, it displays an icon on the screen; if the user chooses to view the page, the WebTV box fetches it over the Internet.

For both of these systems the producer of a program (or commercial) chooses relevant documents by hand. In fact, the producer often creates new documents specifically to be accessed by TV viewers.

In this paper we study the problem of *automatically* finding news articles on the web relevant to the ongoing stream of TV *broadcast news*. We restrict our attention to broadcast news since it is very popular and information-oriented (as supposed to entertainment-oriented).

Our approach is to extract queries from the ongoing stream of closed captions, issue the queries in real time to a news search engine on the web, and postprocess the top results to determine the news articles that we show to the user. We evaluated a variety of algorithms for this problem, looking at the impact of inverse document frequency, stemming, compounds, history, and query length on the relevance and coverage of news articles returned in real time during a broadcast. We also evaluated several postprocessing techniques for improving the precision, including reranking using additional terms, reranking by document similarity, and filtering on document similarity. The best algorithm achieves a precision of 91% on one data set and 84% on a second data set and finds a relevant article for at least 70% of the topics in the data sets.

In general, we find that it is more important to concentrate on a good postprocessing step than on a good query generation step. The difference in precision between the best and the worst query generation algorithm is at most 10 percentage points, while our best postprocessing step improves precision by 20 percentage points or more. To offset the impact of postprocessing on the total number of relevant articles retrieved, we simply increased the number of queries.

To be precise, the best algorithm uses a combination of techniques. Our evaluation indicates that the most important features for its success are a "history feature" and a postprocessing step that filters out irrelevant articles. Many of the other features that we added to improve the query generation do not have a clearly beneficial impact on precision. The "history feature" enables the algorithm to consider all terms since the start of the current topic when generating a query. It tries to detect when a topic changes and maintains a data structure that represents all terms in the current topic, weighted by age. The filtering step discards articles that seem too dissimilar to each other or too dissimilar to the current topic. We also experimented with other postprocessing techniques but they had only a slight impact on precision.

Our algorithms are basically trying to extract keywords from a stream of text so that the keywords represent the "current" piece of the text. Using existing terminology this can be called *time-based keyword extraction*. There is a large body of research on topic detection and text summarization. Recently, time-based summarization has also been studied [1], but to the best of our knowledge there is no prior work on time-based keyword extraction.

The remainder of this paper is organized as follows. Section 2 describes the different query generation algorithms and the different postprocessing steps. Section 3 presents the evaluation. Section 4 discusses related work. We conclude in Section 5.

2. Our approach

Our approach to finding articles that are related to a stream of text is to create queries based on the text and to issue the queries to a search engine. Then we postprocess the answers

returned to find the most relevant ones. In our case the text consists of closed captioning of TV news, and we are looking for relevant news articles on the web. Thus we issue the queries to a news search engine.

We first describe the algorithms we use to create queries and then the techniques we use for postprocessing the answers.

2.1. Query generation

It is an interesting question how best to present the articles found by the system to the users. However, in this article we concentrate on the algorithms for finding relevant articles and do not address user interface issues. Thus, we simply assume that we are interested in showing relevant articles at a regular rate during the news broadcast. As a result the query generation algorithm needs to issue a query periodically, i.e., every s seconds. It cannot wait for the end of a topic. We chose $s = 15$ for two reasons. (1) Empirically we determined that showing an article every 10–15 seconds allows the user to read the title and scan the first paragraph. The actual user interface may allow the user to pause and read the current article more thoroughly. (2) A caption text of 15 seconds corresponds to roughly three sentences or roughly 50 words. This should be enough text to generate a well-specified query.

Because postprocessing may eliminate some of the candidate articles, we return two articles for each query. We also tested at $s = 7$, thus allowing up to half of the candidate articles to be discarded while maintaining the same or better coverage as $s = 15$.

The query generation algorithm is given the *text segment* T since the last query generation. It also keeps information about the previous stream of text. We consider seven different query generation algorithms, described in the following sections. All but the last query generation algorithm issue 2-term queries. A *term* is either a word or a 2-word compound like *New York*. Two-term queries are used because experiments on a test set (different from the evaluation set used in this paper) showed that 1-term queries are too vague and return many irrelevant results. On the other hand, roughly half of the time 3-term queries are too specific and do not return any results (because we are requiring all terms to appear in the search results). The last query generation algorithm uses a combination of 3- and 2-term queries to explore whether the 2-term limit hurts performance.

As is common in the information retrieval (IR) literature [19] the *inverse document frequency idf* of a term is a function of the frequency f of the term in the collection (i.e., the number of documents in the collection containing the term) and the number N of documents in the collection. Specifically, we use the function $\log(N/(f+1))$. Since we do not have a large amount of closed caption data available, we used Google's web collection to compute the *idf* of the terms. This means N was over 2 billion, and f was the frequency of a term in this collection. Unfortunately, there is a difference in word use in written web pages and spoken TV broadcasts. As a result we built a small set of words that are common in captions but rare in the web data. Examples of such words are *reporter* and *analyst*. All of the algorithms below ignore the terms on this stopword list.

The query generation algorithms as well as the postprocessing algorithms that we describe below can process the text in real-time.

2.1.1. The baseline algorithm A1-BASE Our baseline algorithm is a simple $tf \cdot idf$ based algorithm. It uses only single-word terms, not two-word compound terms. Each term is weighted by $tf \cdot idf$, where tf is the frequency of the term in the text segment T. This results in larger weights for terms that appear more frequently in T, and larger weights for more unusual terms. This is useful since doing a search with the more distinctive terms of the news story is more likely to find articles related to the story. The baseline algorithm returns the two terms with largest weight as the query.

2.1.2. The $tf \cdot idf^2$ algorithm A2-IDF2 This is the same algorithm as the baseline algorithm, but a term is weighted by $tf \cdot idf^2$. The motivation is that rare words, like named entities, are particularly important for issuing focussed queries. Thus, the idf component is more important than tf.

2.1.3. The simple stemming algorithm A3-STEM In the previous two algorithms each term is assigned a weight. Algorithm A3-STEM assigns instead a weight to each *stem*. The *stem* of a word is approximated by taking the first 5 letters of the word. For example, *congress* and *congressional* would share the same stem, *congr*. The intention is to aggregate the weight of terms that describe the same entity. We use this simple method of determining stems instead of a more precise method because our algorithm must be real-time.

For each stem we store all the terms that generated the stem and their weight. The weight of a term is $c \cdot tf \cdot idf^2$, where $c = 1$ if the term was a noun and $c = 0.5$, otherwise. (Nouns are determined using the publicly available Brill tagger [2].) We use this weighting scheme since nouns are often more useful in queries than other parts of speech. The weight of a stem is the sum of the weights of its terms.

To issue a query the algorithm determines the two top-weighted stems and finds the top-weighted term for each of these stems. These two terms form the query.

2.1.4. The stemming algorithm with compounds, algorithm A4-COMP Algorithm A4-COMP consists of algorithm A3-STEM extended by two-word compounds. Specifically, we build stems not only for one-word terms, but also for two-word compounds. For this we use a list of allowed compounds compiled from Google's corpus of web data. Stems are computed by stemming both words in the compound, i.e., the stem for the compound *veterans administration* is *veter-admin*. Compounds are considered to be terms and are weighted as before. Queries are issued as for algorithm A3-STEM, i.e., it finds the top-weighted term for the two top-weighted stems. Since a term can now consists of a two-word compound, a query can now, in fact, consist of two, three, or four words.

2.1.5. The history algorithm A5-HIST Algorithm A5-HIST is algorithm A4-COMP with a "history feature." All previous algorithms generated the query terms solely on the basis of the text segment T that was read since the last query generation. Algorithm A5-HIST uses terms from previous text segments to aid in generating a query for the current text segment, the notion being that the context leading up to the current text may contain terms that are still valuable in generating the query.

It does this by keeping a data structure, called the *stem vector*, which represents the previously seen text, i.e., the history. It combines this information with the information produced by algorithm A4-COMP for the current text segment T and finds the top weighted stems.

To be precise, for each stem the stem vector keeps a weight and a list of terms that generated the stem, each with its individual weight. The stem vector keeps the stems of all words that were seen between the last reset and the current text segment. A *reset* simply sets the stem vector to be the empty vector; it occurs when the topic in a text segment changes substantially from the previous text segment (see below).

When algorithm A5-HIST receives text segment T it builds a second stem vector for it using algorithm A4-COMP. Then it checks how similar T is to the text represented in the old stem vector by computing a similarity score *sim*. To do this we keep a stem vector for each of the last three text segments. (Each text segment consists of the text between two query generations, i.e., it consists of the text of the last s seconds.) We add these vectors and compute the dot-product of this sum with the vector for T, only considering the weights of the terms and ignoring the weights of the stems. If the similarity score is above a threshold a_1, then T is *similar* to the earlier text. If the similarity score is above a_2 but below a_1, then T is *somewhat similar* to the earlier text. Otherwise T is *dissimilar* from the earlier text.

If text segment T is similar to the earlier text, the old stem vector is *aged* by multiplying every weight by 0.9 and then the two vectors are added. To add the two vectors, both vectors are expanded to have the same stems by suitably adding stems of weight 0. Also the set of terms stored for each stem is expanded to consist of the same set by adding terms of weight 0. Then the two vectors are added by adding the corresponding weights of the stems and of the terms.

If text segment T is very dissimilar from the earlier text, then the old stem vector is reset and is replaced by the new stem vector. To put it another way, when the current text is very different than the previous text, it means that the topic has changed, so previous history should be discarded in deciding what query to issue.

If text segment T is somewhat similar to the earlier text, then the stem vector is not reset, but the weights in the old stem vector are decreased by multiplying them with a weight that decreases with the similarity score *sim*. Afterwards the old stem vector and the new stem vector are added. So even though the topic has not completely changed, previous terms are given less weight to allow for topic drift.

We used a test data set (different from the evaluation data sets) to choose values for a_1 and a_2 in the *sim* calculation. In our implementation, $a_1 = 0.001$ and $a_2 = 0.0003$. When T is somewhat similar, we use the weight multiplier $a = 0.9^{2-1000 \cdot sim}$, which was chosen so that $a \leqslant 0.9$, i.e., the weights are more decreased than in the case that T is similar to the early text.

In the resulting stem vector the top two terms are found in the same way as in algorithm A4-COMP.

2.1.6. The query shortening algorithm A6-3W To verify our choice of query length 2 we experimented with a query shortening algorithm, which issues a multiple term query,

and shortens the query until results are returned from the news search engine. Earlier experiments showed that reducing the query to one term hurt precision. Therefore we kept two terms as the minimum query length. The query shortening algorithm A6-3W is identical to A5-HIST, but begins with three-term queries, reissuing the query with the two top-weighted terms if there are no results.

2.1.7. Algorithm A7-IDF Algorithm A7-IDF is identical to algorithm A5-HIST with idf^2 replaced by idf. (Note that each increasing algorithm A1–A6 adds one additional feature to the previous. A7-IDF does not fit this pattern; we created it in order to test the specific contribution of idf^2 to A5-HIST's performance.)

2.2. Postprocessing

After generating the search queries we issue them to a news search engine and retrieve the top at most 15 results. Note that each result contains exactly one news article. Because we want to retrieve articles that are about the current news item, we restricted the search to articles published on the day of the broadcast or the day before.

We applied several ways of improving upon these search results, described in the sections below, and then selected the top two results to show to the user as news articles related to the broadcast news story.

Since several queries will be issued on the same topic, they may yield similar result sets and many identical or near identical articles may end up being shown to the user. In fact, in the data sets used for the evaluation (see Section 3.1), queried at both $s = 7$ and $s = 15$, an average of 40% of articles returned would be near-duplicates. Such a large number of duplicates would lead to a poor user experience, so we employed a near-duplicate backoff strategy across all the algorithms. If an article is deemed a near-duplicate of one that has already been presented, the next article in the ranking is selected. If all articles in the result set are exhausted in this manner, the first article in the result set is returned (even though it was deemed a near-duplicate). This reduces the number of repeated highly similar articles to an average of 14% in the evaluation data sets.

To detect near-duplicates without spending time fetching each article, we look at the titles and summaries of the articles returned by the search engine. We compare these titles and summaries to a cache of article titles and summaries that have already been displayed during the broadcast. A similarity metric of more than 20% word overlap in the title, or more than 30% word overlap in the summary, was successful in identifying exact matches (e.g., the same article returned in the results for a different query) and slight variants of the same article, which are commonly issued by news wires as the story develops over time.

The postprocessing steps we used were boosting, similarity reranking, and filtering.

2.2.1. Boosting The news search engine gets a two-term query and does not know anything else about the stream of text. The idea behind boosting is to use additional high-weighted terms to select from the search results the most relevant articles. To implement this idea the query generation algorithm returns along with the query associated *boost terms*

and *boost values*. The boost terms are simply the top five terms found in the same way as the query terms. The boost values are the IDF values of these terms.

The boosting algorithm then reranks the results returned from the search by computing a weight for each result using the boost terms. For a boost term which has IDF *idf* and occurs *tf* times in the text summary returned with the result, the weight is incremented by the value $idf \cdot 4tf/(tf+3)$, which is a $tf \cdot idf$-like formula that limits the influence of the *tf* part to 4. For boost terms in the title, the weight is increased by twice that value. Finally, to favor more recent articles, the weight is divided by $d+1$, where d is the number of days since the article was published. Since we restrict articles to the current date and the day before, the weight is divided by either 1 or 2. The results are then reordered according to their weight; non-boosted results or ties are kept in their original order.

2.2.2. Similarity reranking A second way of reranking is to compute for each of the results returned by the search engine its similarity to the text segment T and to rerank the search results according to the similarity score. To implement this idea we build a $tf \cdot idf$-weighted term vector for both the text segment T and the text of the article and compute the normalized cosine similarity score. (The first 500 characters of the article are used.) This filtering step requires first fetching the articles, which can be time-expensive.

2.2.3. Filtering The idea behind filtering is to discard articles that are very dissimilar to the closed caption text. Additionally, when the issued query is too vague, then the top two search results often are very dissimilar. (Indeed, all the results returned by vague queries are often very different from one another.) So whenever we find two candidate articles and they are dissimilar, we suspect a vague query and irrelevant results. So we discard each of the articles unless it is itself highly similar to the caption.

We again used the $tf \cdot idf$-weighted term vector for the text segment T and the text of the article and computed the normalized cosine similarity score as in the similarity reranking, above. Whenever the page-T similarity score is below a threshold b the article is discarded (*Rule F1*). If there are two search results we compute their similarity score and discard the articles if the score is below a threshold p (*Rule F2*)—but allowing each article to be retained if its page-T similarity score is above a threshold g (*Rule F3*).

We analyzed a test data set (different from the evaluation data sets) to determine appropriate thresholds. In our implementation, $b = 0.1$, $g = 0.3$, and $p = 0.35$.

3. Evaluation

To evaluate different algorithms on the same data set the evaluators worked off-line to give them ample time for their decisions. They were supplied with two browser windows. One browser window contained the article to be evaluated. The article was annotated with an input box so that the score for the article could simply be input into the box. The other browser window contained the part of the closed caption text for which the article was generated. The evaluators were instructed as follows:

You will be reading a transcript of a television news broadcast. What you will be evaluating will be the relevance of articles that we provide periodically during the broadcast. For each displayed article consider whether the article is relevant to at least one of the topics being discussed in the newscast for this article. Use the following scoring system to decide when an article is relevant to a topic:

- 0, *if the article is not on the topic;*
- 1, *if the article is about the topic in general, but not the exact story;*
- 2, *if the article is about the exact news story that is being discussed.*

For example, if the news story is about the results of the presidential election, then an article about a tax bill in congress would score a 0; *an article about the candidates' stands on the environment would score a* 1; *an article about the winner's victory speech would score a* 2.

Do not worry if two articles seem very similar, or if you've seen the article previously. Just score them normally. The "current topic" of the newscast can be any topic discussed since the last article was seen. So if the article is relevant to any of those topics, score it as relevant. If the article is not relevant to those recent topics, but is relevant to a previous segment of the transcript, it is considered not relevant; give it a 0.

We count an article as "relevant" (R) if it was given a score of 1 or 2 by the human evaluator. We count it as "very relevant" (R+) if it was given a score of 2.

To compare the algorithms we use *precision*, i.e., the percentage of relevant articles out of all returned articles. *Recall* is usually defined as the percentage of returned relevant articles out of all relevant articles that exist. However, this is very hard to measure on the web, since it is very difficult to determine all articles on a given topic. In addition, our algorithms are not designed to return all relevant documents, but instead a steady stream of relevant documents. Thus, we define the *pooled recall* to be the percentage of returned relevant articles out of all relevant articles pooled from *all of the query generation algorithms with all postprocessing variants.* We use pooled recall instead of the number of relevant documents to enable comparison over different data sets. To aid in the comparison we also give the number of returned documents that are relevant.

There are various problems with using pooled recall. (1) The pooled recall numbers are appropriate for comparing performance among the different algorithms, but not useful as an absolute measure of an algorithm's recall performance, since no algorithm would be able to achieve 100% pooled recall. This is because when a query is issued at a text segment, an algorithm is limited to returning a maximum of two articles. Since each topic lasts only a fixed duraction of time, only a limited number of articles can be shown. (2) The pooled recall numbers cannot be used to compare an algorithm that issues a search every 7 seconds with and algorithm that issues a search every 15 seconds as the earlier algorithm shows about twice as many articles.

Since pooled recall is a problematic metric for measuring "coverage" we also measure *topic coverage*, which is the percentage of topics (defined below) that have at least one relevant article. The experiments showed a strong correlation between topic coverage and pooled recall, indicating that pooled recall is still a usefull metric.

To understand the relationship of the different algorithms we compute their overlap, both in terms of issued queries and in terms of articles returned. Since filtering is such a powerful technique we study its effectiveness in more detail.

3.1. Data sets

We evaluated all these approaches using the following two data sets:

(1) **HN:** three 30-minute sessions of CNN Headline News, each taken from a different day, and
(2) **CNN:** one hour of Wolf Blitzer Reports on CNN from one day and 30 minutes from another day.

The Headline News sessions ("HN") consists of many, relatively short, news stories. The Wolf Blitzer Reports ("CNN") consists of fewer news stories discussed for a longer time and in greater depth.

Both data sets contain *news stories* and *meta-text*. Meta-text consists of the text between news stories, like "and now to you Tom" or "thank you very much for this report." For evaluating the performance of our algorithms we manually decomposed the news stories into *topics*, ignoring all the meta-text. (This manual segmentation is not an input to the algorithms; it was used strictly for evaluation purposes.) Each topic consists of at least 3 sentences on the same theme; we do not count 1–2 sentence long "teasers" for upcoming stories as topics. The shortest topic in our data sets is 10 seconds long, the longest is 426 seconds long. The average length of a topic in the HN data set is 51 seconds and the median is 27 seconds. The topics comprise a total of 4181 seconds (70 minutes) out of the 90-minute-long caption stream. In the CNN data set the average topic length is 107 seconds and the median is 49 seconds. The topics comprise a total of 3854 seconds (64 minutes).

3.2. Evaluation of the query generation algorithms

3.2.1. Evaluation of the algorithms We first evaluated all the baseline algorithms with two different ways of postprocessing, namely no postprocessing and postprocessing by both boosting and filtering. The CNN data set consists of 3854 seconds, and thus an algorithm that issues a query every 15 seconds issues 257 queries. We return the top two articles for each query so that a maximum of 514 relevant articles could be returned for this data set when $s = 15$. For the HN data set the corresponding number is 557.

The pool of all relevant documents found by any of the algorithms for the HN data set is 846, and for the CNN data set is 816. Thus the pooled recall for each algorithm is calculated by dividing the number of relevant documents it found by these numbers. Note that for $s = 15$ no algorithm can return more than 557 (for HN) or 514 (for CNN) relevant articles, so in those cases the maximum possible pooled recall would be $557/846 = 66\%$ (HN) or $514/816 = 63\%$ (CNN).

Table 1. HN data set: precision p, pooled recall r, and number n of returned documents that are relevant.

		Postprocessing					
		None			Boost + filter		
Technique	s	p	r	n	p	r	n
A1-BASE	7	58%	37%	315	86%	31%	264
A2-IDF2	7	58%	37%	316	87%	31%	266
A3-STEM	7	64%	32%	273	88%	29%	246
A4-COMP	7	64%	32%	271	88%	28%	239
A5-HIST	7	64%	36%	303	**91%**	30%	257
A6-3W	7	72%	33%	279	89%	28%	234
A7-IDF	7	61%	38%	322	89%	31%	262
A1-BASE	15	63%	20%	165	**91%**	17%	147
A2-IDF2	15	62%	20%	165	**91%**	18%	152
A3-STEM	15	69%	25%	210	88%	24%	200
A4-COMP	15	70%	26%	221	90%	25%	210
A5-HIST	15	67%	26%	219	89%	24%	201
A6-3W	15	**75%**	24%	201	**91%**	22%	187
A7-IDF	15	59%	26%	223	**91%**	24%	207

Table 2. CNN data set: precision p, pooled recall r, and number n of returned documents that are relevant.

		Postprocessing					
		None			Boost + filter		
Technique	s	p	r	n	p	r	n
A1-BASE	7	43%	27%	221	77%	21%	171
A2-IDF2	7	46%	27%	220	75%	18%	150
A3-STEM	7	43%	23%	185	76%	18%	143
A4-COMP	7	44%	23%	186	76%	17%	140
A5-HIST	7	55%	32%	259	84%	23%	190
A6-3W	7	60%	30%	246	86%	23%	190
A7-IDF	7	52%	25%	206	82%	23%	186
A1-BASE	15	48%	17%	140	83%	14%	116
A2-IDF2	15	60%	16%	134	85%	13%	108
A3-STEM	15	54%	17%	139	76%	14%	113
A4-COMP	15	59%	18%	144	82%	15%	120
A5-HIST	15	61%	25%	200	**88%**	20%	164
A6-3W	15	**71%**	23%	189	83%	21%	164
A7-IDF	15	56%	25%	202	82%	21%	172

Table 1 presents the precision and pooled recall for all the different query generation algorithms for the HN data set. Table 2 shows the corresponding numbers for the CNN data set. These tables lead to a few observations:

(1) All algorithms perform statistically significantly[1] better with a p-value of < 0.003 when postprocessed with boosting and filtering than without postprocessing. Depend-

ing on the algorithm the postprocessing seems to increase the precision by 20–35 percentage points.

(2) For both data sets the highest precision numbers are achieved with postprocessing and $s = 15$. However, the largest pooled recall is achieved without postprocessing and with $s = 7$. This is no surprise: filtering reduces not only the number of non-relevant articles that are returned, but also the number of relevant ones. The impact of postprocessing on the number of relevant articles that are returned varies greatly between algorithms. The maximum change is 71 articles (A1-BASE with $s = 7$ on HN), and the minimum change is 10 articles (A3-STEM with $s = 7$ on HN). Also, reducing s increases the number of queries issued and thus one expects the number of returned articles to increase, both the relevant ones as well as the non-relevant ones. Thus pooled recall increases as well.

(3) Precision on the CNN data set is lower than precision on the HN data set. This is somewhat surprising as longer topics might be expected to lead to higher precision. The reason is that since we issue more queries on the same topic, we reach further down in the result sets to avoid duplicates and end up returning less appropriate articles.

(4) Algorithm A5-HIST with $s = 7$ and with postprocessing performs well in both precision and pooled recall. For the HN data set, it achieves a precision of 91% with 257 relevant articles returned, for the CNN data set it achieves a precision of 84% with 190 relevant articles returned. This means it returns a relevant article every 16 seconds and every 20 seconds, respectively, on the average. The performance of algorithm A6-3W is very similar to algorithm A5-HIST. None of the other algorithms achieves precision of at least 90% and pooled recall of at least 30%. For example, algorithms A1-BASE and A2-IDF2 with $s = 15$ have precision 91% on the HN data set but they return roughly 100 articles fewer that A5-HIST with $s = 7$, which corresponds to a drop of pooled recall by 13 percentage points (A1-BASE) and 12 percentage points (A2-IDF2).

Without postprocessing the difference in precision between A5-HIST and algorithms A1-BASE, A2-IDF2, A3-STEM, and A4-COMP is statistically significant on the CNN data set for $s = 7$. For $s = 15$ the difference between A5-HIST and A1-BASE is significant with a p-value of < 0.004.

(5) Without postprocessing the precision of the baseline algorithm A1-BASE is statistically significantly worse than most of the other algorithms on the CNN data set. Also algorithm A6-3W is statistically significantly better than most of the other algorithms. However, these differences disappear or are no longer statistically significant when filtering and boosting is applied.

3.2.2. Evaluation of the individual techniques We also discuss the contribution of different query generation techniques.

(1) *idf versus idf*^{2}. The baseline algorithm A1-BASE and algorithm A2-IDF2 differ only in the use of idf^2 versus idf. For $s = 15$ and no postprocessing, A2-IDF2 gives a statistically significant improvement over A1-BASE on the CNN data set. In all the other cases their performance is very similar.

Algorithms A5-HIST and A7-IDF also differ only in the use of idf^2 versus idf. Without postprocessing A5-HIST outperforms A7-IDF in precision on both data sets. The differences are statistically significant for $s = 7$ on the CNN data set and for $s = 15$ on the HN data set. With postprocessing their performance is either very similar or the difference is not statistically significant. Altogether, idf^2 seems to work slightly better than idf.

(2) *Stemming*. Adding stemming to algorithm A2-IDF2 gives algorithm A3-STEM. On the HN data set stemming gives an improvement without postprocessing but with postprocessing and $s = 15$ stemming gives slightly worse performance. On the CNN data set stemming hurts precision. This is not so surprising as stemming is mostly used to improve recall. It does increase pooled recall over A3-STEM for $s = 15$, but it has no positive impact or even a negative impact on pooled recall for $s = 7$. Overall, our experiments do not show any benefit that suggests using stemming.

(3) *Compounds*. Algorithm A4-COMP consists of algorithm A3-STEM with 2-word compounding added, i.e., we only evaluated compounding for algorithms that use stemming. Their performance is very similar. The precision of A4-COMP is greater than the precision of A3-STEM for $s = 15$ on the CNN data set but it is not statistically significant. However, for $s = 15$ and no postprocessing, A4-COMP gives a statistically significant improvement (p-value < 0.02) over A1-BASE on the CNN data set. Overall, adding compounds does not seem to significantly improve precision. It has basically no impact on pooled recall.

(4) *History*. Adding a "history feature" to algorithm A4-COMP gives algorithm A5-HIST. The history gives a small improvement in precision for $s = 7$ on the HN data set, while it seems to slightly hurt for $s = 15$. On the CNN data set, A5-HIST clearly outperforms A4-COMP, both in precision and in pooled recall; the difference is statistically significant with p-value < 0.004 for $s = 7$ and no postprocessing. It also noticeably increases pooled recall.

This is not surprising. For longer topics (as the CNN data set has) it becomes valuable to have a history feature, especially if queries are issued every 7 seconds. Each text segment may not on its own contain highly relevant text that can be used as a query in finding similar stories. Shorter text segments suffer even more from this problem. The history rectifies this by effectively extending the length of the text segment in a time-aged manner.

For example, for one of the data sets three shootings were in the news: one in Arizona, one in Oklahoma, and one in Jordan. The algorithms without history sometimes returned non-relevant articles about shootings different than the one being discussed in the broadcast because the current text segment did not mention the location. Algorithm A5-HIST never made this mistake. Altogether, we recommend adding a history feature to a query generation algorithm.

(5) *Query shortening*. Algorithm A6-3W first issues a three-word query and "backs off" to a two-word query if no results were found. This happens for about 60% of the queries. Without postprocessing, its precision is statistically significantly better than all of the other algorithms with $s = 15$ on the CNN data set and for most of the other algorithms for $s = 7$ and also for the HN data set. With boosting and filtering A6-3W

is very similar to algorithm A5-HIST. Pooled recall decreases slightly when compared to A5-HIST. The reason is that three-word queries might return only one result where two-word query would return at least two results. Altogether, trying out three-word queries is helpful without postprocessing, but with postprocessing it does not lead to an improvement.

In conclusion, postprocessing and the "history feature" give the largest improvement in search precision, namely 20–35 percentage points for postprocessing and about 5 percentage points for history. Postprocessing reduces pooled recall by about 6 percentage points, while the history feature has negligible effect on pooled recall. Thus, a query generation algorithm should have both a way to include the history and a postprocessing step that filters out irrelevant documents. None of the other features seem clearly beneficial.

3.2.3. R+ relevance Tables 3 and 4 give the "R+ precision," i.e., percentage of articles exactly on topic (R+: given a score of 2 by the evaluator). They also present the "R+ pooled recall," i.e., the percentage of articles with score R+ out of all articles with score R+ together with the actual number of such articles found by each algorithm. They confirm the above observations. On the HN data set across all algorithms about 80% of the articles rated relevant are rated R+. This number is pretty constant with a low of 73% for A4-COMP, $s = 15$, and no postprocessing and a high of 89% for A1-BASE, $s = 15$, and boosting and filtering on the HN-data set. On the CNN data set about 70% of the articles rated relevant are rated R+.

3.2.4. Precision versus recall As the data above shows postprocessing with boosting and filtering clearly improves precision. It also decreases pooled recall somewhat. Figure 1

Table 3. HN data set: percentage p of articles with score R+ out of all returned articles, percentage r of articles with score R+ out of all articles with score R+, and the number of such returned articles.

		Postprocessing					
		None			Boost + filter		
Technique	s	p	r	n	p	r	n
A1-BASE	7	44%	28%	240	69%	25%	211
A2-IDF2	7	45%	29%	245	70%	25%	214
A3-STEM	7	49%	25%	209	73%	24%	204
A4-COMP	7	50%	25%	209	72%	23%	196
A5-HIST	7	47%	26%	222	76%	25%	214
A6-3W	7	56%	26%	216	75%	23%	197
A7-IDF	7	46%	29%	244	74%	26%	219
A1-BASE	15	53%	16%	139	**81%**	15%	131
A2-IDF2	15	51%	16%	136	78%	15%	131
A3-STEM	15	54%	20%	165	75%	20%	169
A4-COMP	15	51%	19%	162	72%	20%	169
A5-HIST	15	52%	20%	168	71%	19%	159
A6-3W	15	**59%**	19%	158	75%	18%	155
A7-IDF	15	46%	20%	173	75%	20%	171

Table 4. CNN data set: percentage p of articles with score R+ out of all returned articles, percentage r of articles with score R+ out of all articles with score R+, and the number of such returned articles.

		Postprocessing					
		None			Boost + filter		
Technique	s	p	r	n	p	r	n
A1-BASE	7	30%	19%	155	61%	16%	134
A2-IDF2	7	31%	18%	148	59%	14%	117
A3-STEM	7	31%	16%	133	59%	14%	112
A4-COMP	7	31%	16%	130	59%	13%	109
A5-HIST	7	36%	21%	169	64%	18%	145
A6-3W	7	40%	20%	161	61%	17%	136
A7-IDF	7	37%	18%	146	65%	18%	148
A1-BASE	15	35%	12%	101	66%	11%	92
A2-IDF2	15	43%	12%	96	**67%**	10%	85
A3-STEM	15	37%	12%	95	51%	9%	76
A4-COMP	15	39%	12%	95	58%	10%	85
A5-HIST	15	40%	16%	129	60%	14%	112
A6-3W	15	**49%**	16%	130	59%	15%	120
A7-IDF	15	36%	16%	129	56%	14%	118

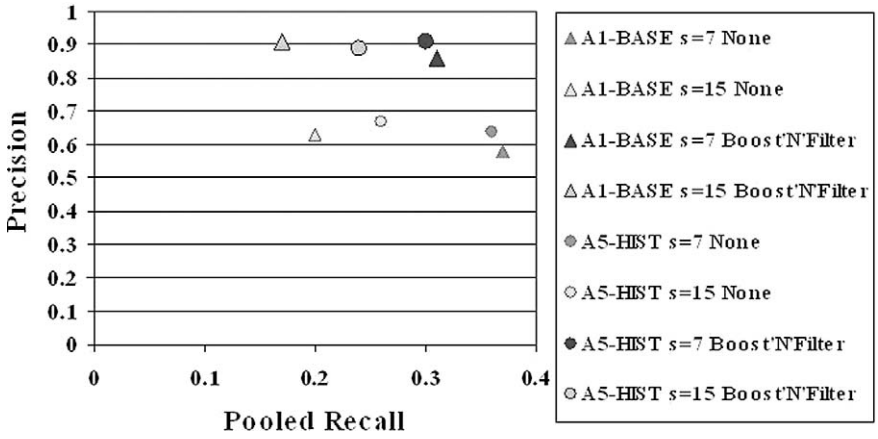

Figure 1. HN data set: precision versus pooled recall for the A1-BASE and the A5-HIST algorithm. Smaller points show the performance of the algorithm without postprocessing, larger points show it with boosting and filtering.

shows the corresponding precision recall graph for A1-BASE for the HN data set. The two entries with solid color both issue queries every 7 seconds, the smaller one has no postprocessing and the larger one uses both boosting and filtering. The entries with shaded color show the corresponding data points when a query is issued every 15 seconds. In both cases the decrease in pooled recall is small while the increase in precision is large. It is also quite obvious that the version with $s = 7$ dominates the version with $s = 15$.

Figure 1 also shows the data for A5-HIST. Smaller points show the performance of the algorithm without postprocessing, larger points show it with boosting and filtering. The clear separation between the lower four entries, which are small, and the higher four entries, which are large, shows the positive impact on precision by the boosting and filtering. The two circles with solid color correspond to algorithm A5-HIST, while the two triangles with solid color belong to algorithm A1-BASE. This shows that for $s = 15$, A5-HIST gives a clear improvement over A1-BASE. The corresponding points with solid color ($s = 7$) are most closer together, indicating that the difference is much smaller for $s = 7$.

3.3. Postprocessing

As we saw in the previous section postprocessing using boosting and filtering gives a big improvement in precision without decreasing pooled recall much. The obvious question is what contributed most to the improvement, boosting or filtering. A second question is whether postprocessing by similarity reranking performs better than postprocessing by boosting.

Since the improvement was unanimous among algorithms and data sets, we evaluated only the HN data set for the full set of combinations of algorithms. Table 5 shows the details for relevance and Table 6 gives the details for the score R+.

For the CNN data set we only evaluated the 3 "most promising" postprocessing techniques, namely boosting together with filtering, similarity reranking, and similiarity reranking together with filtering. The results for relevance are in Table 7, the results for the score R+ are in Table 8.

On the HN data set the improvement is clearly achieved by the filtering step, the boosting step is only giving a small improvement. All of the differences between boosting alone and filtering and boosting are statistically significant. Also, all of the differences between

Table 5. HN data set: precision and pooled recall in parenthesis.

		Postprocessing					
Technique	*s*	None	Boost	Filter	Boost + filter	Sim. rerank	Sim. rerank + filter
A2-IDF2	7	58% (37%)	58% (37%)	88% (32%)	87% (31%)	60% (38%)	84% (34%)
A4-COMP	7	64% (32%)	66% (33%)	86% (27%)	88% (28%)	68% (34%)	86% (32%)
A5-HIST	7	64% (36%)	64% (36%)	91% (29%)	91% (30%)	64% (36%)	88% (31%)
A2-IDF2	15	62% (20%)	64% (20%)	89% (17%)	91% (18%)	66% (21%)	92% (20%)
A4-COMP	15	70% (26%)	72% (27%)	93% (23%)	90% (25%)	74% (27%)	91% (25%)
A5-HIST	15	67% (26%)	69% (26%)	92% (22%)	89% (24%)	71% (26%)	92% (25%)

Table 6. HN data set: percentage of returned articles with score R+ out of all returned documents, and percentage of returned articles with score R+ out of all articles with score R+.

		Postprocessing					
Technique	*s*	None	Boost	Filter	Boost + filter	Sim. rerank	Sim. rerank + filter
A2-IDF2	7	45% (29%)	46% (29%)	73% (27%)	70% (25%)	49% (31%)	70% (29%)
A4-COMP	7	50% (25%)	52% (26%)	70% (22%)	72% (23%)	56% (28%)	72% (27%)
A5-HIST	7	47% (26%)	49% (28%)	71% (23%)	76% (25%)	50% (28%)	72% (25%)
A2-IDF2	15	51% (16%)	54% (17%)	76% (14%)	78% (15%)	57% (18%)	81% (17%)
A4-COMP	15	51% (19%)	55% (21%)	74% (18%)	72% (20%)	58% (21%)	74% (20%)
A5-HIST	15	52% (20%)	52% (20%)	75% (18%)	71% (19%)	57% (21%)	76% (20%)

Table 7. CNN data set: precision, pooled recall in parenthesis, and the number of returned articles that are relevant.

		Postprocessing			
Technique	*s*	None	Boost + filter	Sim. rerank	Sim. rerank + filter
A2-IDF2	7	46% (27%)	75% (18%)	48% (29%)	75% (23%)
A4-COMP	7	44% (23%)	76% (17%)	47% (25%)	77% (20%)
A5-HIST	7	55% (32%)	84% (23%)	56% (31%)	85% (20%)
A2-IDF2	15	60% (16%)	85% (13%)	59% (17%)	78% (14%)
A4-COMP	15	59% (18%)	82% (15%)	59% (18%)	78% (15%)
A5-HIST	15	61% (25%)	88% (20%)	65% (23%)	81% (20%)

boosting alone and filtering alone are statistically significant. In some cases filtering alone gives even higher precision than filtering and boosting together.

Similarity reranking seems to give a slightly higher gain in precision than boosting. However, combined with filtering it does not yield better precision than boosting and filtering combined. None of the differences between boosting alone and similarity reranking alone and between boosting with filtering and similarity reranking with filtering are statistically significant.

Note, however, that similarity reranking and filtering together always has better pooled recall than boosting and filtering, which in turn has better pooled recall than filtering alone.

Table 8. CNN data set: percentage of returned articles with score R+ out of all returned documents, and percentage of returned articles with score R+ out of all articles with score R+.

		Postprocessing			
Technique	s	None	Boost + filter	Sim. rerank	Sim. rerank + filter
A2-IDF2	7	31% (18%)	59% (14%)	34% (20%)	56% (17%)
A4-COMP	7	31% (16%)	59% (13%)	33% (17%)	58% (15%)
A5-HIST	7	36% (21%)	64% (18%)	38% (21%)	64% (15%)
A2-IDF2	15	43% (12%)	67% (10%)	45% (13%)	65% (12%)
A4-COMP	15	39% (12%)	58% (10%)	39% (12%)	54% (11%)
A5-HIST	15	40% (16%)	60% (14%)	44% (16%)	60% (15%)

Table 9. HN data set with $s = 7$: percentage of queries that are identical when sorted lexicographically.

	A1-BASE	A2-IDF2	A3-STEM	A4-COMP	A5-HIST	A6-3W	A7-IDF
A1-BASE		94%	27%	25%	10%	6%	10%
A2-IDF2	94%		30%	27%	12%	7%	10%
A3-STEM	27%	30%		87%	31%	19%	28%
A4-COMP	25%	27%	87%		38%	19%	34%
A5-HIST	10%	12%	31%	38%		40%	63%
A6-3W	6%	7%	19%	19%	40%		30%
A7-IDF	10%	10%	28%	34%	63%	30%	

To summarize, filtering gives a large precision improvement: about 20–30 percentage points with a decrease of 6 percentage points in pooled recall. Filtering and similarity reranking together achieve the same precision but return roughly 10% more relevant articles than filtering alone.

3.4. Query overlap and URL overlap

Given a postprocessing step the performance of the different query selection algorithms is very similar. An obvious question to ask is whether the reason for this similarity is that the algorithms issue very similar queries. To answer this question we compute the similarity between the queries issued by the different query selection algorithms, i.e., we compare the ith query issued by one algorithm with the ith query issued by another algorithm. Table 9 gives the percentage of queries that have identical terms (though not necessarily ordered identically) for $s = 7$ and the HN data set. Note that we are looking at all generated queries, i.e., the queries *before* the postprocessing step.

Table 10. HN data set with $s = 7$: percentage of URLs of algorithm A that are also returned by algorithm B, where the choice of A determines the row and the choice of B determines the column. Since different algorithms return a different number of URLs the table is not symmetric.

	A1-BASE	A2-IDF2	A3-STEM	A4-COMP	A5-HIST	A6-3W	A7-IDF
A1-BASE		93%	36%	33%	15%	11%	13%
A2-IDF2	96%		37%	36%	17%	13%	15%
A3-STEM	41%	41%		83%	36%	23%	21%
A4-COMP	36%	38%	80%		42%	24%	28%
A5-HIST	16%	18%	35%	42%		39%	40%
A6-3W	13%	15%	23%	26%	43%		38%
A7-IDF	15%	17%	22%	30%	43%	38%	

Table 11. HN data set: percentage of topics with at least one relevant article and percentage of topics with at least one article rated R+.

		Score R		Score R+	
Technique	s	None	Boost + filter	None	Boost + filter
A1-BASE	7	78%	73%	76%	70%
A2-IDF2	7	79%	76%	76%	72%
A3-STEM	7	74%	70%	70%	67%
A4-COMP	7	76%	72%	70%	68%
A5-HIST	7	77%	70%	73%	67%
A6-3W	7	73%	70%	70%	68%
A7-IDF	7	73%	73%	72%	70%
A1-BASE	15	63%	59%	60%	56%
A2-IDF2	15	63%	61%	60%	60%
A3-STEM	15	72%	67%	70%	67%
A4-COMP	15	76%	72%	73%	71%
A5-HIST	15	72%	65%	68%	65%
A6-3W	15	71%	66%	66%	63%
A7-IDF	15	71%	69%	70%	63%

The table shows that nearly all queries are identical for related algorithms like A1-BASE and A2-IDF2. However, for algorithms A1-BASE and A5-HIST, for example, only 10% of the queries are identical.

Even if the queries are quite different, there could still be a large overlap in the URLs returned at a given point in the stream of text. However, that is also not the case as Table 10 shows for the HN data set and $s = 7$. The results for $s = 15$ are similar.

To summarize, the overlap both in queries and in articles is high between A1-BASE and A2-IDF2 and is high between A3-STEM and A4-COMP but is low otherwise. Thus, even though the algorithms have similar performance when used with postprocessing, it is in general not due to the same queries being issued or the same URLs being returned. It might, hence, be possible to improve precision by combining the algorithms in the right way.

3.5. Topic coverage

Another question to ask is how many of the topics receive at least one relevant article. In the HN data set there were a total of 82 topics. In Table 11 we show the percentage

Table 12. CNN data set: percentage of topics with at least one relevant article and percentage of topics with at least one article rated R+.

		Score R		Score R+	
Technique	s	None	Boost + filter	None	Boost + filter
A1-BASE	7	86%	81%	83%	81%
A2-IDF2	7	83%	81%	81%	75%
A3-STEM	7	83%	72%	72%	69%
A4-COMP	7	83%	75%	78%	69%
A5-HIST	7	89%	72%	81%	72%
A7-IDF	7	92%	69%	78%	67%
A1-BASE	15	81%	78%	72%	72%
A2-IDF2	15	75%	72%	67%	61%
A3-STEM	15	69%	64%	64%	61%
A4-COMP	15	72%	67%	64%	64%
A5-HIST	15	78%	75%	69%	67%
A7-IDF	15	78%	75%	64%	69%

of topics with at least one relevant article for the HN data set and also the percentage of topics with at least one article rated R+ for the HN data set. Not surprisingly, these percentages are strongly correlated with pooled recall. They are the highest for $s = 7$ with no postprocessing and the lowest for $s = 15$ with postprocessing. It is interesting to note that the numbers are not much lower for the percentage of topics with score R+ than for score R. Said differently, if a topic has a relevant article it most likely also has a topic rated R+.

Table 12 gives the corresponding percentages for the CNN data set. The values are higher as we would expect since the topics are longer. However, there is also more variation in these numbers as there are only 36 topics in the CNN data set.

Figure 2 plots precision versus topic coverage for the HN data sets and algorithms A1-BASE and A5-HIST. The smaller four points correspond to the algorithms without postprocessing, the larger four points to the algorithms with boosting and filtering. As the figure shows all the smaller points are lower than the larger points, i.e., boosting and filtering gives a clear precision improvement with a slight decrease in topic coverage. The data points with shaded color correspond to the algorithms where a query is issued every 15 seconds, the data points with solid color to every 7 seconds. The entries with solid color lie to the right of the corresponding shaded entries, meaning that they achieve higher topic coverage, in some cases with slightly lower precision. As in the precision–recall graphs, both algorithm A1-BASE and algorithm A5-HIST with $s = 7$ and boosting and filtering seem to be good design choices.

We also analyzed longer and shorter topics. Both are equally well covered, i.e., the length is not the distinguishing factor for whether a topic is covered or not. Instead there seem to be topics for which it is "hard" to find relevant articles and others for which it easy. For example, it is easy to find articles for Winona Ryder's shoplifting trial: her name is rare and thus had high *idf*, and she is not mentioned in other news for that day. For other topics it is hard to find related news stories, mostly because they fall into the

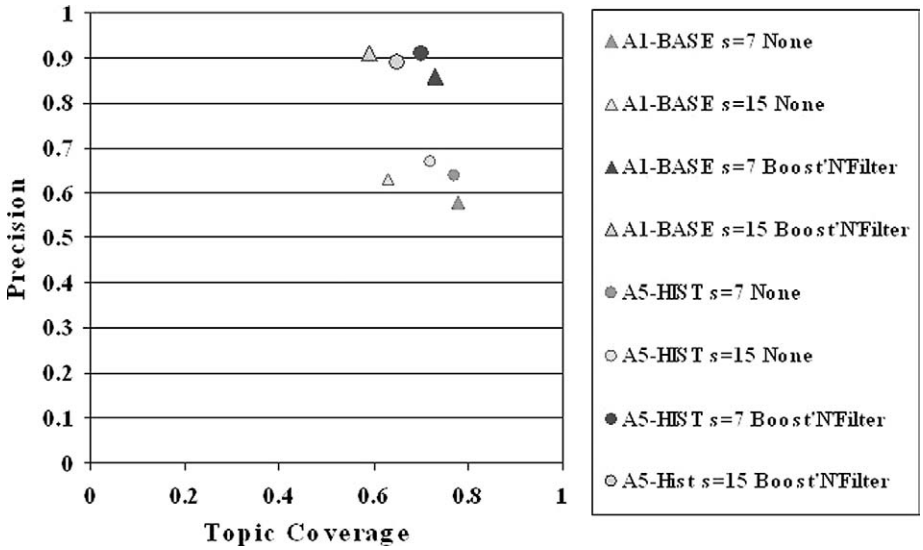

Figure 2. HN data set: precision versus topic coverage for the A1-BASE and the A5-HIST algorithms. Smaller points show the performance of the algorithm without postprocessing, larger points show it with boosting and filtering.

category of "unusual" news. Examples include a story about a beauty pageant for women in Lithuania's prisons, a story about a new invention that uses recycled water from showers and baths to flush toilets, and a story about garbage trucks giving English lessons over loudspeakers in Singapore.

In summary, roughly 70% of the topics have at least one article rated relevant, and almost as many have at least one article rated very relevant (R+). The length of the topic does not seem to be a factor in determining whether a relevant article can be found for it.

3.6. Filtering effectiveness

Filtering is very powerful in improving precision. It consists of three rules:

Rule F1. Whenever the page-T similarity score is below a threshold b the article is discarded.
Rule F2. If there are two search results we compute their similarity score and discard the articles if the score is below a threshold p.
Rule F3. If Rule F2 triggers we check each of the two articles. If its page-T similarity score is above a threshold g it is retained, where $g > b$.

Note that it is possible for multiple rules to apply.

When evaluated alone, similarity reranking achieved higher precision than boosting, while similarity reranking with filtering performed about the same as the other combinations with filtering. Thus, filtering in combination with similarity reranking led to the least

Table 13. HN data set: for each filtering rule the percentage of filtered articles that are filtered by the technique. The percentages for a given algorithm can add up to over 100% since both filtering rules can apply.

Technique	s	# returned articles	# filtered articles	% filtered by F1	% filtered by F2
A2-IDF2	7	534	172	22%	91%
A4-COMP	7	424	107	9%	93%
A5-HIST	7	468	157	32%	92%
A2-IDF2	15	264	81	16%	89%
A4-COMP	15	311	75	19%	85%
A5-HIST	15	316	83	19%	88%

Table 14. CNN data set: for each filtering rule the percentage of filtered articles that are filtered by the technique. The percentages for a given algorithm can add up to over 100% since both filtering rules can apply.

Technique	s	# returned articles	# filtered articles	% filtered by F1	% filtered by F2
A2-IDF2	7	483	204	19%	92%
A4-COMP	7	427	197	15%	95%
A5-HIST	7	446	179	23%	91%
A2-IDF2	15	232	80	17%	95%
A4-COMP	15	247	78	12%	92%
A5-HIST	15	295	83	14%	94%

precision improvements, i.e., performed worst. Since we expect filtering to perform better with no other postprocessing or with boosting, we decided to evaluate in this section filtering with similarity reranking.

Rules F1 and F2 both filter out articles and we analyzed which of the two rules filters out more articles. Tables 13 and 14 show the percentage of articles that each filtering rule filtered. The percentage can add up to over 100% since both rules can apply. It clearly shows that F2 filters out most of the articles.

We also wanted to evaluate for each filtering rule how often it makes the wrong decision. For F1 and F2 this means that they discard a relevant article. Rule F3 makes the wrong decision if it keeps an irrelevant article that F2 would have discarded. Tables 15 and 16 gives the error rate ρ for each filtering rule and also the number f of articles that were filtered (for F1 and F2) or retained (for F3) by the rule. For F1 and F2, the *error rate* is the percentage of relevant articles out of all articles filtered by the rule. For F3, it is the percentage of irrelevant articles out of all articles whose similarity with text T is above the threshold g and which F2 would have discarded.

Rules F1 and F2 usually have error rates around 20%. Only for algorithm A5-HIST on the CNN data set does F1 have an error rate of 50%, but in this case F1 filtered only 12 articles. Rule F3 kicks in very infrequently, but is usually wrong. However, since the number of articles retained by F3 is so small its impact on the overall precision is small.

Table 15. HN data set: for each filtering rule the error rate ρ and the number of articles that were filtered (for F1 and F2) or retained (for F3) by the rule.

		F1		F2		F3	
Technique	s	ρ	f	ρ	f	ρ	f
A2-IDF2	7	14%	37	17%	156	50%	10
A4-COMP	7	10%	10	17%	100	56%	9
A5-HIST	7	18%	50	19%	145	33%	4
A2-IDF2	15	0%	13	13%	72	91%	11
A4-COMP	15	0%	14	27%	64	78%	2
A5-HIST	15	6%	16	16%	73	78%	9

Table 16. CNN data set: for each filtering rule the error rate ρ and the number of articles that were filtered (for F1 and F2) or retained (for F3) by the rule.

		F1		F2		F3	
Technique	s	ρ	f	ρ	f	ρ	f
A2-IDF2	7	18%	38	17%	188	80%	5
A4-COMP	7	7%	30	16%	188	100%	5
A5-HIST	7	23%	40	19%	162	100%	4
A2-IDF2	15	17%	6	26%	76	100%	2
A4-COMP	15	22%	9	22%	72	50%	2
A5-HIST	15	50%	12	27%	78	0%	1

The conclusion that can be drawn in this section is that Rule F2 filters out almost all articles and has an error rate of roughly 20%. Rule F1 is not needed as it only adds a small number of additional filtered articles and Rule F3 should be avoided as it has a high error rate.

4. Related work

4.1. Query-free search

To our knowledge, there has been little work on automatically selecting documents that a user might want to see while watching a TV program. However, there is a significant literature on the broader problem of query-free information retrieval: finding documents that are relevant to a user's current activity, without requiring an explicit query. The various systems differ in what stream of text they consider as input and what genre of related documents they return. We will use the "Input–Output" notation below.

Broadcast news–web pages. The Cronkite system [11], developed around the same time as our system, has a goal similar to ours: present relevant information culled from databases and the web during a television news broadcast. Cronkite segments the news stream into stories using topic markers found in the closed captioning as well as cue phrases spoken by the newscasters. It can classify the story into one of four topics (e.g., Middle East,

medical) and then prepares queries based on the story text and on named entities mentioned in the story. Topic- and entity-specific information, such as maps of a country in the Middle East or stock price graphs for companies, is retrieved and organized into linked web pages along with results from web news search engines.

Web pages–web pages. The Letizia system [10] observes a user browsing the web, and suggests other web pages the user may find interesting. Rather than searching an index of web pages, it "surfs ahead" of the user, following hyperlinks from the page the user is currently viewing. Similarly, commercial browser assistants such as Autonomy Kenjin and PurpleYogi (both no longer available) suggest related web pages based on the content of web pages the user has been viewing.

Problem report–repair manual. Another early query-free IR system is FIXIT [8], which helps technicians as they use an expert system to diagnose and repair copiers. FIXIT identifies the currently reported symptoms and the faults it considers likely, then maps these symptoms and faults to keywords, and retrieves sections of the copier documentation that match these words.

User behavior–personal files. The just-in-time IR project at MIT [15,16] has focused on retrieving personal files—such as notes and archived email messages—that a user would currently find useful. This project first produced the Remembrance Agent, which looks at a document the user is editing in Emacs and matches fragments of this document (such as the last 50 words) against a corpus of personal files. The follow-up Margin Notes system performs a similar task, but observes the web pages that a user views in a web browser. Finally, the Jimminy system runs on a wearable computer. Jimminy bases its suggestions on what the user is reading or writing on the heads-up display, as well as on Global Positioning System data and active badge data indicating what other people are nearby. All these systems use a common information retrieval backend based on the Okapi similarity metric [17].

The XLibris pen-based document reader [14] allows users to mark up documents as they are reading. The system derives queries from the passages of text that are marked, and searches over a local corpus for relevant documents to present to the user.

User behavior–news and stock quotes. The SUITOR system [12] tracks user behavior like what applications are running and what text the user is currently writing to build a model of the user's current interest. It uses this model to find information that is interesting to the user, such as news headlines and stock quotes.

Open documents in editor or browser–web pages. Another system similar in purpose to our own is Watson [4], which suggests web pages to a computer user based on the documents currently open in a word processor or web browser. Watson uses a variety of heuristics to construct queries from the text of the documents, then sends these queries to the AltaVista search engine.

Email–web pages. Our work is also related to a small prototype system that constructed queries from email messages and sent them to an early version of the Google search engine [3].

4.2. Text summarization and keyword extraction

In the Information Retrieval literature there has been a plethora of work on topic detection and text summarization. Recently, the problem of time-based summarization has been studied. See [1] for an excellent overview of the area. Our work is different in two ways:

(1) It does not need to identify topics; it only needs to detect whether the current topic is different from the previous topic. If a later topic is very similar to a topic discussed much earlier, the system does not need to recognize this.
(2) The system does not need to construct a summary; it extracts keyphrases that can be used to formulate a search query.

The research on keyphrase extraction, see, e.g., [7,9,13,20], and specifically the algorithm by Turney [21], is the most related to our work. The main difference to our work is that we study the time-based variant of the problem, which also includes topic change detection.

5. Conclusion

This paper evaluated seven algorithms and three postprocessing techniques for finding news articles on the web relevant to news broadcasts. For this genre of television show, the best algorithm (A5-HIST with $s = 7$, boosting, and filtering) finds a relevant page every 16–20 seconds on average, achieves a precision of 84–91%, and finds a relevant article for about 70% of the topics. Our experiments clearly show that filtering articles by similarity to the caption text and similarity with each other gives a large improvement in precision. It would be interesting future work to refine and improve upon the filtering technique presented in this paper. Without postprocessing algorithm A2-IDF2 together with history and query shortening looks very promissing. It would be interesting to evaluate this combination. It would also be interesting to experiment with different ways of using the history for query generation.

The news search engine we used restricted us to using Boolean retrieval. It is an interesting open question whether a weighted term-vector retrieval would have improved the search quality sufficiently to make post-filtering redundant.

The framework of the system is not limited to news, however; we have considered simple methods of detecting other genres (such as sports, weather, and "general" topics) and sending such queries to appropriate web information sources. The genres could be identified by using machine learning on a labelled corpus of television captions; an even simpler way would be to use television schedules and their associated metadata to categorize the current show into a genre.

Finally, as voice recognition systems improve, the same kind of topic finding and query generation algorithms described in this paper could be applied to conversations, providing relevant information immediately upon demand.

Acknowledgements

We would like to thank Shahid Choudhry for providing us with closed caption transcripts for our experiments.

Note

1. To determine statistical significance we used the rank-sum test and the t-test. If a p-value is given, it is the p-value of the rank-sum test, as it is more conservative. If statistical significance is claimed but no p-value is given, the p-value of the rank-sum test is less than 0.05.

References

[1] J. Allan, R. Gupta, and V. Khandelwal, "Temporal summaries of news topics," in *Research and Development in Information Retrieval*, 2001, pp. 10–18.
[2] E. Brill, "Transformation-based error-driven learning and natural language processing: A case study in part-of-speech tagging," *Computation Linguistics* 21(4), 1995, 543–565.
[3] S. Brin, R. Motwani, L. Page, and T. Winograd, "What can you do with a web in your pocket?" *Data Engineering Bulletin* 21(2), 1998, 37–47.
[4] J. Budzik, K. Hammond, and L. Birnbaum, "Information access in context," *Knowledge Based Systems* 14(1–2), 2001, 37–53.
[5] J. Davis, "Intercast dying of neglect," *CNET News*, January 29, 1997.
[6] Electronic Industries Alliance, "Transport of internet uniform resource locator (url) information using text-2 (t-2) service," Technical Report, EIA-746-A, 1998.
[7] E. Frank, G. W. Paynter, I. H. Witten, C. Gutwin, and C. G. Nevill-Manning, "Domain-specific keyphrase extraction," in *IJCAI*, 1999, pp. 668–673.
[8] P. Hart and J. Graham, "Query-free information retrieval," *IEEE Expert* 12(5), 1997, 32–37.
[9] B. Krulwich and C. Burkey, "Learning user information interests through the extraction of semantically significant phrases," in *AAAI 1996 Spring Symposium on Machine Learning in Information Access*, 1996.
[10] H. Lieberman, "Letizia: An agent that assists web browsing," in C. S. Mellish (ed.), *Proceedings of the 14th International Joint Conference on Artificial Intelligence (IJCAI-95)*, 1995, pp. 924–929.
[11] K. Livingston, M. Dredze, K. Hammond, and L. Birnbaum, "Beyond broadcast," in *International Conference on Intelligent User Interfaces*, 2003.
[12] P. Maglio, R. Barrett, C. Campbell, and T. Selker, "Suitor: An attentive information system," in *International Conference on Intelligent User Interfaces*, 2000.
[13] A. Munoz, "Compound key word generation from document databases using a hierarchical clustering art model," *Intelligent Data Analysis* 1(1), 1997.
[14] M. N. Price, G. Golovchinsky, and B. N. Schilit, "Linking by inking: Trailblazing in a paper-like hypertext," in *Proceedings of the Hypertext'98*, 1998, pp. 30–39.
[15] B. Rhodes and P. Maes, "Just-in-time information retrieval agents," *IBM Systems Journal* 39(3–4), 2000.
[16] B. J. Rhodes, "Just-in-time information retrieval," Ph.D. Thesis, MIT Media Laboratory, Cambridge, MA, May 2000.
[17] S. Robertson, S. Walker, and M. Beaulieu, "Okapi at TREC-7: automatic ad hoc, filtering, VLC and interactive track," in *Proceedings of the 7th International Text Retrieval Conference (TREC)*, 1999, pp. 253–264.

[18] G. D. Robson, "Closed captions, V-chip, and other VBI data," *Nuts and Volts*, 2000.
[19] G. Salton, *The SMART System—Experiments in Automatic Document Processing*, Prentice Hall, 1971.
[20] A. M. Steier and R. K. Belew, "Exporting phrases: A statistical analysis of topical language," in *Proceedings of the 2nd Symposium on Document Analysis and Information Retrieval*, 1993, pp. 179–190.
[21] P. D. Turney, "Learning algorithms for keyphrase extraction," *Information Retrieval* 2(4), 2000, 303–336.